Understanding Neural Networks via Rule Extraction

Rudy Setiono and **Huan Liu** *
Department of Information Systems and Computer Science
National University of Singapore
Kent Ridge, Singapore 0511
{rudys,liuh}@iscs.nus.sg

Abstract

Although backpropagation neural networks generally predict better than decision trees do for pattern classification problems, they are often regarded as black boxes, i.e., their predictions are not as interpretable as those of decision trees. This paper argues that this is because there has been no proper technique that enables us to do so. With an algorithm that can extract rules[1], by drawing parallels with those of decision trees, we show that the predictions of a network can be explained via rules extracted from it, thereby, the network can be understood. Experiments demonstrate that rules extracted from neural networks are comparable with those of decision trees in terms of predictive accuracy, number of rules and average number of conditions for a rule; they preserve high predictive accuracy of original networks.

1 Introduction

Researchers [Dietterich *et al.*, 1990; Quinlan, 1994; Shavlik *et al.*, 1991] have compared experimentally the performance of learning algorithms of decision trees and neural networks (NNs). A general picture of these comparisons is that: (1) Backpropagation (an NN learning method) usually requires a great deal more computation; (2) the predictive accuracy of both approaches is roughly the same, with backpropagation often slightly more accurate [Quinlan, 1994]; and (3) symbolic learning (decision trees induction) can produce interpretable rules while networks of weights are harder to interpret [Shavlik *et al.*, 1991]. In effect, a neural network is widely regarded as a black box due to the fact that little is known about how its prediction is made.

Our view is that this is because we are not equipped with proper techniques to know more about how a neural network makes a prediction. If we can extract rules from neural networks as generating rules from decision trees, we can certainly understand better how a prediction is made. In addition, rules are a form of knowledge that can be easily verified by experts, passed on and expanded. Some recent works [Fu, 1994; Saito and Nakano, 1988; Towell and Shavlik, 1993] have shown that rules can be extracted from networks. These algorithms are search-based methods that have exponential complexity. Subsets of incoming weights that exceed the bias on a unit are searched. Such sets are then rewritten as rules. To simplify the search process, some assumptions are made. One assumption is that the activation of a unit is either very close to 1 or very close to 0. This can restrict the capability of the network since when the sigmoid transfer function is used as the the activation function, the activation of a unit can have any value in the interval (0,1).

In this paper, a novel way to understand a neural network is proposed. Understanding a neural network is achieved by extracting rules with a three-phase algorithm: first, a weight-decay backpropagation network is built so that important connections are reflected by their bigger weights; second, the network is pruned such that insignificant connections are deleted while its predictive accuracy is still maintained; and last, rules are extracted by recursively discretizing the hidden unit activation values. By drawing parallels with the rules generated from decision trees, we show that networks can be interpreted by the extracted rules; the rules in general preserve the accuracy of the networks; and they also explain how a prediction is made.

2 A Three-Phase Algorithm

A standard three layer feedforward network is the base of the algorithm. Weight decay is implemented while backpropagation is carried out. After the network is pruned, its hidden units activation values are discretized. Rules are extracted by examining the discretized activation values of the hidden units. The algorithm is described in steps below.

2.1 Backpropagation with Weight Decay

The basic structure of the neural network in this work is a standard three-layer feedforward network, which consists of an input layer, I, a hidden layer, H, and an output

*Published at IJCAI'95.

[1]Rules are in forms of "if $x_1 = v(x_1)$ and $x_2 = v(x_2)$ ... and $x_n = v(x_n)$ then C_j" where x_i's are the inputs to the network, $v(x_i)$'s are one of the values x_i can have, and C_j is the network's prediction.

layer, O. The number of input units corresponds to the dimensionality of the examples of a classification problem. The number of output units is determined by the number of classes in the data. The number of hidden units depends on the problem in hand. Two approaches to determine a suitable number of hidden units have been described in the literature. The first approach begins with a minimal number of hidden units, one or two, and more hidden units are added as they are needed to increase the accuracy of the network. The second approach begins with an oversized network and removes redundant connections in the network by pruning. In the process, hidden units that are not connected to any input units or/and output units can be removed as well. We adopt the second approach since we are also interested in removing irrelevant input units.

Given an n-dimensional example x^i, $i \in$ {1,2,...,k} as input, let w_l^m be the weight for the connection from input unit l, $l \in$ {1,2,...,n} to hidden unit m, $m \in$ {1,2,...,h} and v_p^m be the weight from hidden unit m to output unit p, $p \in$ {1,2,...,o}, the pth output of the network for example x^i is obtained by computing

$$S_p^i = \sigma(\sum_{m=1}^{h} \alpha^m v_p^m), \quad \text{where} \tag{1}$$

$$\alpha^m = \delta(\sum_{l=1}^{n} x_l^i w_l^m), \quad \delta(x) = (e^x - e^{-x})/(e^x + e^{-x}). \tag{2}$$

The target output for an example x^i that belongs to class C_j is an o-dimensional vector t^i, where $t_p^i = 0$ if $p \neq j$ and $t_j^i = 1$, $j, p = 1, 2, \ldots o$. The backpropagation algorithm is applied to update the weights (w, v) and minimize the following function:

$$\theta(w, v) = F(w, v) + P(w, v),$$

where $F(w, v)$ is the cross entropy function:

$$F(w, v) = -\sum_{i=1}^{k}\sum_{p=1}^{o}\left(t_p^i \log S_p^i - (1 - t_p^i)\log(1 - S_p^i)\right).$$

and $P(w, v)$ is a penalty term used for weight decay:

$$P(w, v) = \epsilon_1 \left(\sum_{m=1}^{h}\sum_{\ell=1}^{n} \frac{\beta(w_\ell^m)^2}{1 + \beta(w_\ell^m)^2} + \sum_{m=1}^{h}\sum_{p=1}^{o} \frac{\beta(v_p^m)^2}{1 + \beta(v_p^m)^2}\right)$$
$$+ \; \epsilon_2 \left(\sum_{m=1}^{h}\sum_{\ell=1}^{n} (w_\ell^m)^2 + \sum_{m=1}^{h}\sum_{p=1}^{o} (v_p^m)^2\right).$$

Using this penalty function, we have been able to obtain networks with less number of connections than the previously reported for many learning problems. Details of the comparison and the choice of the parameter values (ϵ_1, ϵ_2, and β) are given in [Setiono, 1995].

2.2 Network Pruning

A network pruning algorithm is briefly described below.

Neural network pruning algorithm

1. Let η_1 and η_2 be positive scalars such that $\eta_1 + \eta_2 < 0.5$.
2. Pick a fully connected network. Train this network until a predetermined accuracy rate is achieved and for each correctly classified example the condition
$$\max_p |e_p^i| = \max_p |S_p^i - t_p^i| \leq \eta_1, \tag{3}$$
where $\eta_1 \in [0, 0.5)$, is satisfied.
3. Let (w, v) be the weights of this network.
For each w_ℓ^m, if
$$\max_p |v_p^m w_\ell^m| \leq 4\eta_2, \tag{4}$$
then remove w_ℓ^m from the network
4. For each v^m, if
$$\max_p |v_p^m| \leq 4\eta_2, \tag{5}$$
then remove v_p^m from the network
5. If no weight satisfies condition (4) or condition (5), then remove w_ℓ^m with the smallest product $\max_p |v_p^m w_\ell^m|$.
6. Retrain the network. If classification rate of the network falls below an acceptable level, then stop. Otherwise, go to Step 3.

This pruning algorithm removes the connections of the network according to the magnitudes of their weights (4 and 5). As our eventual goal is to get a set of simple rules that describe the classification process, it is important that all unnecessary connections be removed. In order to remove as many connections as possible, it is therefore imperative that the weights be prevented from taking values that are too large. At the same time, weights of irrelevant connections should be encouraged to converge to zero.

2.3 Rule Extraction

When network pruning is completed, the network contains only those salient connections. Nevertheless, rules are not readily extractable because the hidden unit activation values are continuous. The discretization of these values paves the way for rule extraction. The following algorithm discretizes the activation values (many clustering algorithms can be used for this purpose).

Discretizing Hidden Unit Activation Values by Clustering

For each hidden unit,

1. Let $\epsilon \in (0, 1)$,
2. Start with activation value α_0 of the first example in the training set,
3. Cluster activation values (α_i) for the remaining examples if $|\alpha_i - \alpha_0| < \epsilon$,
4. Represent this cluster's activation value by the average of the activation values in the cluster;
5. Select next α_0, repeat 3 and 4 for clustering until all activation values are clustered.

When the clustering is done, the network's accuracy is checked to see if it drops or not. A very small ϵ can guarantee that the network with discretized activation values is as accurate as the original network with continuous activation values. So if it's accuracy does not drop and there are still many discrete values, clustering can be performed again with a larger ϵ. Otherwise, ϵ should be reduced to a smaller value.

After network pruning and activation value discretization, rules can be extracted by examining the possible combinations in the network outputs (explained in detail in Section 3.2). The actual rule extraction is done by an algorithm that generates 100% accurate rules [Liu, 1995]. However, when there are still too many connections (e.g., more than 7) between a hidden unit and input units, the extracted rules may not be easy to understand. Another three layer feedforward subnetwork may be employed to simplify rule extraction for the hidden unit. This subnetwork is trained in the same ways as is the original network, but in a reduced scale: the number of output units is the number of discrete values of the hidden unit, while the input units are those connected to the hidden unit in the original network. Examples are grouped according to their discretized activation values. Given d discrete activation values $D_1, D_2, \ldots, D_d$, all examples with activation values equal to D_j are given a d-dimensional target value of all zeros except for one 1 in position j. A new hidden layer is introduced for this subnetwork and it is then trained, pruned, and the activation values of its hidden units are discretized for rule extraction. If necessary, another subnetwork is created until the number of connection is small enough or the new subnetwork cannot simplify the connections between the inputs and the hidden unit at the higher level. The creation of subnetworks is rarely needed. For example, in our experiments, it was only used for the Splice-junction problem.

3 Experiments and Results

In this section, we describe the datasets and representations used in experiments. A detailed example is given to show how the three-phase algorithm is applied to extracting rules. Summary of the results on all datasets are given with a comparison to those produced by the decision tree induction methods. Understanding a neural network is achieved by being able to explain, based on the rules, how each prediction is made in parallel with understanding a decision tree by having rules generated from it [Quinlan, 1993].

3.1 Datasets and Representations

Three datasets used are: 1. Iris – a classic dataset introduced by R.A. Fisher [1936]; 2. Breast Cancer – a widely tested real-world dataset for the Wisconsin Breast Cancer diagnosis; and 3. Splice-junction – a dataset used in splice-junction determination originally described by Noordewier et al [1991]. The datasets are obtainable from the University of California Irvine data repository for machine learning (via anonymous ftp from ics.uci.edu). The summary of these datasets, their representations, and how each dataset is used in experiments are given below.

- *Iris* - the dataset contains 50 examples each of the classes Iris setosa, Iris versicolor, and Iris virginica (species of iris). Each example is described using four numeric attributes ($\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$): sepal-length, sepal-width, petal-length, and petal-width. Since each attribute takes a continuous value, the ChiMerge algorithm proposed by Kerber [1992] was reimplemented to discretize attribute values. The thermometer code [Smith, 1993] is used to binarize the discretized values; 16, 9, 7, and 6 inputs (discrete values) for $\mathcal{A}_1$,$\mathcal{A}_2$, $\mathcal{A}_3$, and $\mathcal{A}_4$, respectively. With 1 input for bias, there are total 39 inputs and three outputs. Examples in odd positions in the original dataset form the training set and the rest are for testing as was done in [Fu, 1994].
- *Breast Cancer* - the dataset consists of 699 examples, of which 458 examples are classified as benign, and 241 are malignant. 50% examples of each class were randomly selected (i.e., 229 benign and 121 malignant examples) for training, the rest for testing in the experiments. Each example is described by 9 attributes, each attribute takes an ordinal integer from 1 to 10 (10 values). Due to the ordinal nature, the thermometer code is used again to code each attribute value. Ten inputs correspond to 10 values of each attribute with all 10 inputs *on* representing value 10, the rightmost input *on* for value 1, and the two rightmost inputs *on* for value 2, etc.
- *Splice-junction* - the data set contains 3175 examples[2], approximately 25% are exon/intron boundaries (EI), 25% are intron/exon boundaries (IE), and remaining 50% are neither (N). Each example consists of a 60-nucleotide-long DNA sequence categorized with EI, IE or N. Each of these 60 attributes takes one of the four values: G, T, C or A that are coded as 1000, 0100, 0010, and 0001, respectively. The class values (EI, IE, N) are similarly coded as 100, 010, and 001, respectively.For the results presented here, the training data set consists of 1006 examples while the testing data set consists of all 3175 examples.

3.2 A Detailed Example – Iris Data Classification

This example shows in detail how rules are extracted from a pruned network. In the experiment, 100 fully connected neural networks were used as the starting networks. Each of these networks consists of 39 input units, 3 hidden units and 3 output units. These networks were trained with initial weights that had been randomly generated in the interval $[-1, 1]$. Each of the trained networks was pruned until its accuracy on the training data dropped below 95%. The weights and topology of networks with the smallest number of connections and an accuracy rate of more than 97% were saved for possible rule extraction. The results of these experiments are summarized in Table 1 in which we list the average number of connections in the pruned networks and their

[2]Another 15 examples in the original dataset contain invalid values so these examples are not included in experiment.

Min. network acc.	**95 %**	**97 %**
No. connections	9.26 (2.33)	10.66 (3.06)
Acc. on train data	97.20 (1.45) %	99.03 (0.84) %
Acc. on test data	92.32 (2.49) %	94.55 (2.03) %

Table 1: Average number of connections and accuracy of 100 pruned networks on the training and testing data sets of the Iris problem and their standard deviations.

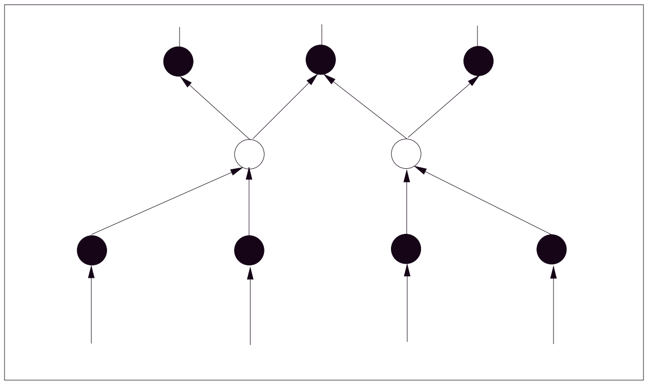

Figure 1: Pruned network with only 8 connections for the Iris problem, accuracy on training set = 98.67 %, accuracy on the testing set = 97.33 %. Figure next to a connection indicates the weight of that connection.

average accuracy rates on the training data and the testing data. Statistics in the second column of this table were obtained from 100 pruned networks, all of which have accuracy rates on the training data of at least 95 %. In the third column, the figures were obtained from 100 pruned networks with accuracy of at least 97 % on the training data.

One of the smallest pruned networks is depicted in Figure 1. It has only 2 hidden units and a total of 8 connections with 98.67% accuracy on the training set and 97.33% on the testing set. We ran the clustering algorithm of Section 2.3 on this network and found only 2 discrete values are needed at each of the two hidden units to maintain the same level of accuracy on the training data. At hidden unit 1, 48 of 75 training examples have activation values equal to 0 and the remaining 27 have activation values equal to 1. At hidden unit 2, the activation value of 25 examples is 1 and the activation value of the remaining 50 examples is -0.5. Since we have two activation values at each of the two hidden units, four different outcomes at the output units are possible (Table 2). From this table, it is clear that an example will be classified as *Iris setosa* as long as its activation value at the second hidden unit is equal to 1. Otherwise, the example is classified as *Iris versicolor* provided that its first hidden unit activation value H_1 = 0. The default class will then be *Iris virginica*.

As seen in Figure 1, only two inputs, I_31 and I_39, determine the activation values of the second hidden unit, H_2. However, since I_39 is 1 for all the training data, H_2 is effectively determined by I_31. Since the weights of the arcs connecting input units 31 and 39 to the second hidden unit are -5.4 and 4.8 respectively, it is easy to conclude that if I_31 = 0, then H_2 is 1, otherwise, H_2 is -0.52. This implies that an example will be classified as *Iris setosa* only if I_31 is 0 (hence H_2 is 1).

Activations		Pred. output			Class
H_1	H_2	O_1	O_2	O_3	
0	1	0.5	0	1	*setosa*
0	-0.5	0.5	1	0	*versicolor*
1	1	0.9	0	1	*setosa*
1	-0.5	0.9	0	0	*virginica*

Table 2: Discretized hidden unit activation values and predicted outputs of the network in Figure 1.

	NN Rules		DT Rules	
	Training	Testing	Training	Testing
setosa	25/25	25/25	25/25	25/25
versicolor	24/25	23/25	24/25	24/25
virginica	25/25	25/25	24/25	22/25
Overall	74/75 98.67%	73/75 97.33%	73/75 97.33%	71/75 94.67%

Table 3: Accuracy of NN rules and DT rules on the training and testing data.

The activation value of the first hidden unit, H_1, depends only on I_26 and I_34. The weights of the arcs connecting input units 26 and 34 to the first hidden unit are 5.1 and 8.1, respectively, hence H_1 is 0 if and only if I_26 = I_34 = 0. Other input combinations will yield value 1 for H_1. Hence, an example with I_31 = 1, I_26 = I_34 = 0 will be classified as *Iris versicolor*.

With the thermometer coding scheme used for the input, a complete set of rules can be easily obtained in terms of the original attributes of the iris data set. The accuracy of this rule set is summarized in Table 3:

NN Rules[3]

Rule 1: If Petal-length $\leq$ 1.9 then *Iris setosa*
Rule 2: If Petal-length $\leq$ 4.9 and Petal-width $\leq$ 1.6 then *Iris versicolor*
Default Rule: *Iris virginica*.

For reference, the rule set (DT Rules) generated by C4.5rules (based on a decision tree method but generate more concise rules than the tree itself) is included here:

DT Rules

Rule 1: If Petal-length $\leq$ 1.9 then *Iris setosa*
Rule 2: If Petal-length $>$ 1.9 and Petal-width $\leq$ 1.6 then *Iris versicolor*
Rule 3: If Petal-width $>$ 1.6 then *Iris virginica*
Default Rule: *Iris setosa*.

3.3 Comparisons

In this section, parallels are drawn between rules extracted from both neural networks and decision trees

[3]The rules are fired in a top-down and left-to-right fashion.

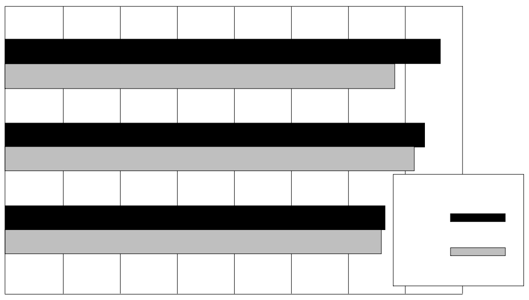

Figure 2: Comparison: predictive accuracy for the three datasets.

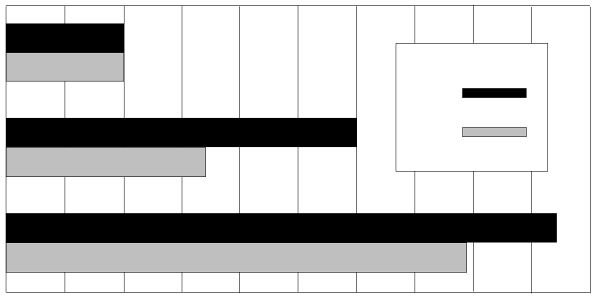

Figure 3: Comparison: average number of a rule's conditions for the three datasets.

(NN rules vs. DT rules). *Understandability* is partly defined as being explicable in the sense that a prediction can be explained in terms of inputs (or attribute values). Choosing to compare NN rules with DT rules is due to the fact that DT rules are considered best understandable among the available choices. A rule in discussion consists of two parts: the if-part is made of a conjunction of conditions, and the then-part specifies a class value. The conditions of a rule are in forms of "A_i=V_j", i.e., attribute A_i takes value V_j. When a rule is fired, a prediction is given that the example under consideration belongs to class C_k. By examining the fired rule, it can be explained how the prediction is attained. If necessary, the intermediate process can also be explicitly explained.

C4.5 and C4.5rules [Quinlan, 1993] were run on the above three datasets to generate DT rules. Briefly, C4.5 generates a decision tree which C4.5rules generalizes to rules. Since researchers [Cheng *et al.*, 1988; Shavlik *et al.*, 1991] observed that mapping many-valued variables to two-valued variables results in decision trees with higher classification accuracy[4], the same binary coded data for neural networks were used for C4.5 and C4.5rules.

Being explicable is only one aspect of understandability. A rule with many conditions is harder to understand than a rule with fewer conditions. Too many rules also hinder humans understanding of the data under examination. In addition to understandability, rules without generalization (i.e., high accuracy on testing data) are not much of use. Hence, the comparison is performed along three dimensions: 1. predictive accuracy; 2. average number of conditions of a rule; and 3. number of rules (see Figures 2-4).

The reasoning behind the comparisons is that if NN rules are comparable with DT rules, since the latter are admittedly interpretable, so should the former. Now that each prediction can be explained in light of some rule, and those rules have direct links to the neural network, it can be concluded that the network's behavior can be understood via those rules.

[4]It's true indeed for the three datasets in our experiments.

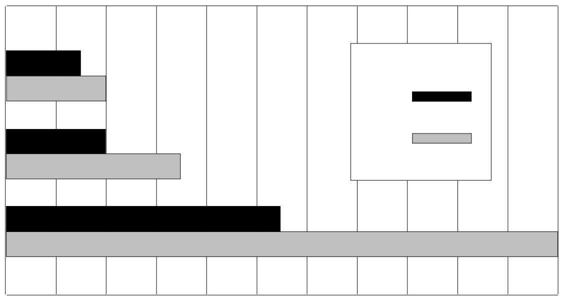

Figure 4: Comparison: number of rules for the three datasets.

4 Discussion

The comparisons made in Figures 2-4 indicate that NN rules are comparable with, if not better than, DT rules in terms of our understanding measures. The average number of conditions in NN rules is higher than that of DT for 2 of the 3 problems tested, however, the total number of NN rules is less than DT rules for all the 3 problems. These observations are consistent with the nature of each learning algorithm, i.e., parallel vs. sequential. Other issues of interests are:

- *The training time.* It takes much longer time to train a neural network than to learn a decision tree. This is also true for NN rules and DT rules extraction. Due to the existence of sequential and parallel data types, and decision trees and neural networks are best suited to one type only [Quinlan, 1994], the two approaches are expected to coexist. When time is really scarce, the decision tree approach should be taken. Otherwise, it is worthwhile trying both because of backpropagation's other advantages (generalizing better on a smaller dataset, predicting better in general, etc. [Towell and Shavlik, 1993]).
- *Average performance of NN rules.* Because of neural networks' nondeterministic nature, it is not uncommon that many runs of networks are needed with

different initial weights. As was shown in Table 1, the average performance for 100 pruned networks is very impressive (94.55%). This displays the robustness of the presented algorithm.

- *Accuracy of neural networks and NN rules.* There is a trade-off between the accuracy of the the rules extracted from the network and the complexity of the rules. A network can be further pruned and simpler rules obtained at the cost of sacrificing its accuracy. A notable feature of our rule extraction algorithm is that while it allows us to extract rules with the same accuracy level as that of the pruned network, it is also possible to simplify the rules by considering a smaller number of hidden unit activation values.
- *Understanding the weights of connections.* Unlike M-of-N rules [Towell and Shavlik, 1993], NN rules here reflect precisely how the network works. NN rules given here are actually the merge of the two sets: 1. from the input layer to the hidden layer; and 2. from the hidden layer to the output layer. NN rules cover all the possible combinations of the connections with various input values and discrete activation values of hidden units. This is a significant improvement over search-based methods [Towell and Shavlik, 1993; Fu, 1994] where all possible input combinations are searched for subsets that will exceed the bias on a unit. To reduce the cost of searching, they normally limit the number of antecedents in extracted rules. Our algorithm imposes no such limit.
- *Consistency between NN and DT rules.* Consistency checking is not an easy task. In general, the possible rule space is very large since the training data is only a sample of the world under consideration. It is not surprising that there exist many equally good rule sets. Using the binary code for the Iris data, for example, the possible size of the rule space is 2^{38}, but there are only 75 examples for training. However, for simple problem like the Iris problem, the rules extracted by NN and the rules generated by DT are remarkably similar.

5 Conclusion

Neural networks have been considered black boxes. In this paper, we propose to understand a network by rules extracted from it. We describe a three-phase algorithm that can extract rules from a *standard* feedforward neural network. Network training and pruning is done via the simple and widely-used backpropagation method. No restriction is imposed on the activation values of the hidden units or output units. Extracted rules are a one-to-one mapping of the network. They are compact and comprehensible, and do not involve any weight values. The accuracy of the rules from a pruned network is as high as the accuracy of the network. Experiments show that NN rules and DT rules are quite comparable. Since DT rules are regarded as explicit and understandable, we conclude that NN rules are likewise. With the rules extracted by the method introduced here, neural networks should no longer be regarded as black boxes.

References

[Cheng *et al.*, 1988] J. Cheng, U.M. Fayyad, K.B. Irani, and Z Qian. Improved decision trees: A generalized version of id3. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 100–106. Morgan Kaufman, 1988.

[Dietterich *et al.*, 1990] T.G. Dietterich, H. Hild, and G. Bakiri. A comparative study of id3 and backpropagation for english text-to-speech mapping. In *Machine Learning: Proceedings of the Seventh International Conference.* University of Texas, Austin, Texas, 1990.

[Fisher, 1936] R.A. Fisher. The use of multiple measurements in taxonomic problems. *Ann. Eugenics*, 7(2):179–188, 1936.

[Fu, 1994] L. Fu. *Neural Networks in Computer Intelligence.* McGraw-Hill, 1994.

[Kerber, 1992] R. Kerber. Chimerge: Discretization of numeric attributes. In *AAAI-92, Proceedings Ninth National Conference on Artificial Intelligence*, pages 123–128. AAAI Press/The MIT Press, 1992.

[Liu, 1995] H. Liu. Generating perfect rules. Technical report, Department of Info Sys and Comp Sci, National University of Singapore, February 1995.

[Noordewieer *et al.*, 1991] M.O. Noordewieer, G.G. Towell, and J.W. Shavlik. Training knowledge-based neural networks to recognize genes in dna sequences. In *Advances in neural information processing systems*, volume 3, pages 530–536. Morgan Kaufmann, 1991.

[Quinlan, 1993] J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[Quinlan, 1994] J.R. Quinlan. Comparing connectionist and symbolic learning methods. In S.J. Hanson, G.A. Drastall, and R.L. Rivest, editors, *Computational Learning Therory and Natural Learning Systems*, volume 1, pages 445–456. A Bradford Book, The MIT Press, 1994.

[Saito and Nakano, 1988] K. Saito and R Nakano. Medical diagnostic expert system based on pdp model. In *Proceedings of IEEE International Conference on Neural Networks*, volume 1, pages 255–262. IEEE, 1988.

[Setiono, 1995] R. Setiono. A penalty function approach for pruning feedforward neural networks. *Technical Report, DISCS, National University of Singapore*, 1995.

[Shavlik *et al.*, 1991] J.W. Shavlik, R.J. Mooney, and G.G. Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6(2):111–143, 1991.

[Smith, 1993] M. Smith. *Neural networks for Statistical Modeling.* Van Nostrand Reinhold, 1993.

[Towell and Shavlik, 1993] G.G. Towell and J.W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101, 1993.